Iris_dataset

May 6, 2021

1 Importamos nuestras dependencias

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# !pip install seaborn
import seaborn as sns
```

In [2]:
```
# !pip install sklearn
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# no tiene relevancia la siguiente dependencia
# se utiliza para visualizar .png que importamos desde el ordenador.
from IPython.core.display import Image, display
```

2 Importamos nuestro set de datos (dataset)

In [3]:
```
# ruta: C:\Users\jmp_e\Desktop\datasets
# Nombre del .csv: 2_IrisSpecies.csv
df = pd.read_csv("C:/Users/jmp_e/Desktop/datasets/2_IrisSpecies.csv")
# imprimimos las primeras 5 líneas
df.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

3 Analizamos brevemente la información que tenemos

In [4]:
```
df.Species.value_counts()
```

Out[4]:
```
Iris-virginica     50
Iris-setosa        50
```

```
Iris-versicolor    50
Name: Species, dtype: int64
```

```
In [5]: # primeras 10 líneas (en este caso), por defecto son 5
        df.head(10)
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
In [6]: # imprimimos las últimas 5 líneas
        df.tail()
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

	Species
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica
149	Iris-virginica

4 Eliminamos 1 columna sin información relevante

```
In [7]: df = df.drop("Id", axis=1)
        # axis = 1 le indicará que por columnas..
        # se podrían borrar más columnas de 1 solo paso de hecho

        # imprimimos las primeras 2 filas
        df.head(2)
```

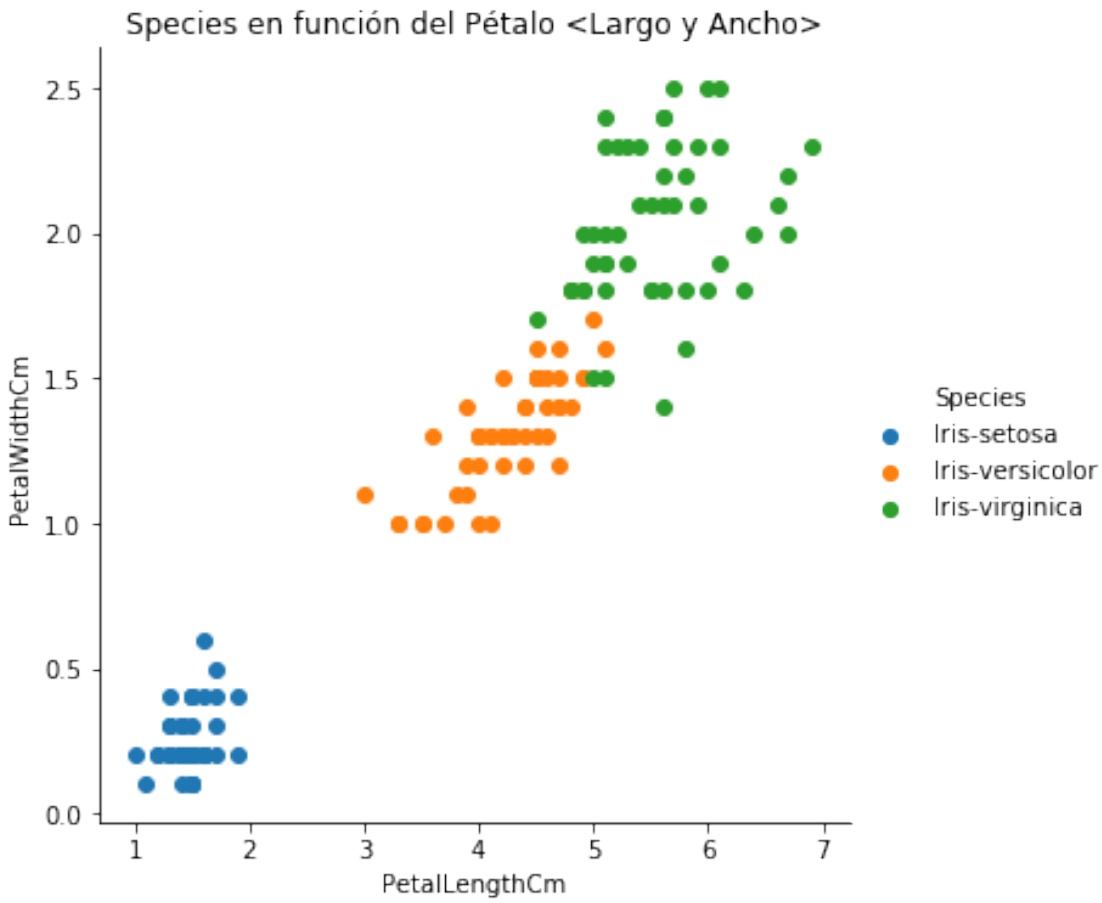
Out[7]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

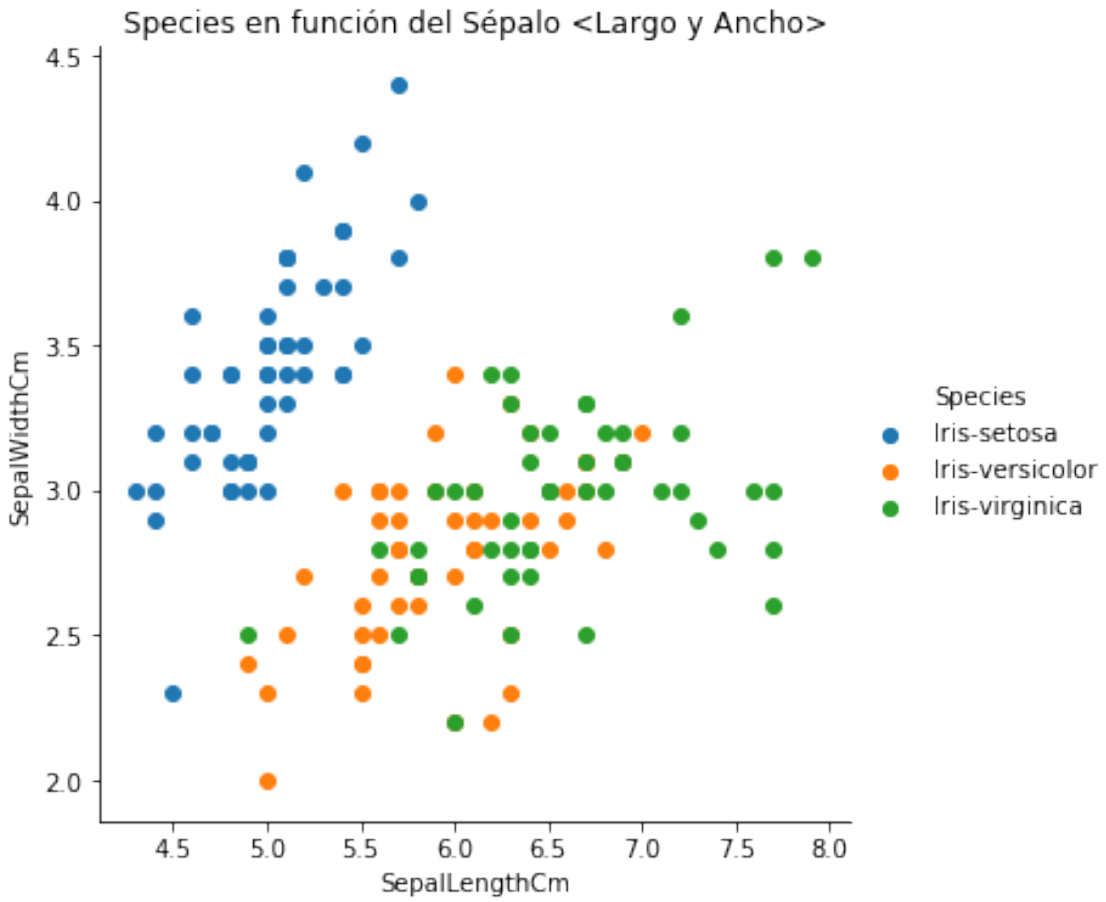
5 Ploteamos algunas curvas (seaborn)

In [8]:
```
# Sobre esto de las gráficas se habla más en Big Data y Machine Learning..
```

In [9]:
```
sns.FacetGrid(df,hue="Species", height=5.2) \
                    .map(plt.scatter, "PetalLengthCm", "PetalWidthCm") \
                    .add_legend()

plt.title("Species en función del Pétalo <Largo y Ancho>")
plt.show()
```



In [10]:
```
sns.FacetGrid(df,hue="Species", height=5.2) \
                    .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
                    .add_legend()

plt.title("Species en función del Sépalo <Largo y Ancho>")
plt.show()
```



6 recordando la información real..

```
In [11]: df.head(2)
```

Out[11]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

7 Analizo la información (columna "Species")

```
In [12]: # NombreDataframe.NombreColumna para "apuntar" a una columna concreta
         df.Species
```

```
Out[12]: 0         Iris-setosa
         1         Iris-setosa
         2         Iris-setosa
         3         Iris-setosa
         4         Iris-setosa
```

```
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

8 Modifico los 3 tipos de Species en 0-1-2

```
In [13]: # Modifico en: 0-1-2 (en este caso) para la salida:
         # Y se lo asigno a la propia columna.

         df.Species = df.Species.map({"Iris-setosa": 0,
                                      "Iris-versicolor": 1,
                                      "Iris-virginica": 2})
         df.head()
```

Out[13]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
In [14]: print(np.array(df.Species.tolist()))
         # que no despiste el np.array() simplemente es para que me lo imprima ordenado.

         # print(df.Species.tolist()) es lo que realmente hacemos..

         # Otra forma de hacer esto sería con un bucle, pero .map es más rápido
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

NOTAS:
Este dataset no requiere de escalado de datos,
no es relevante saber ahora qué es eso,
pero sirve para tratar de evitar que:

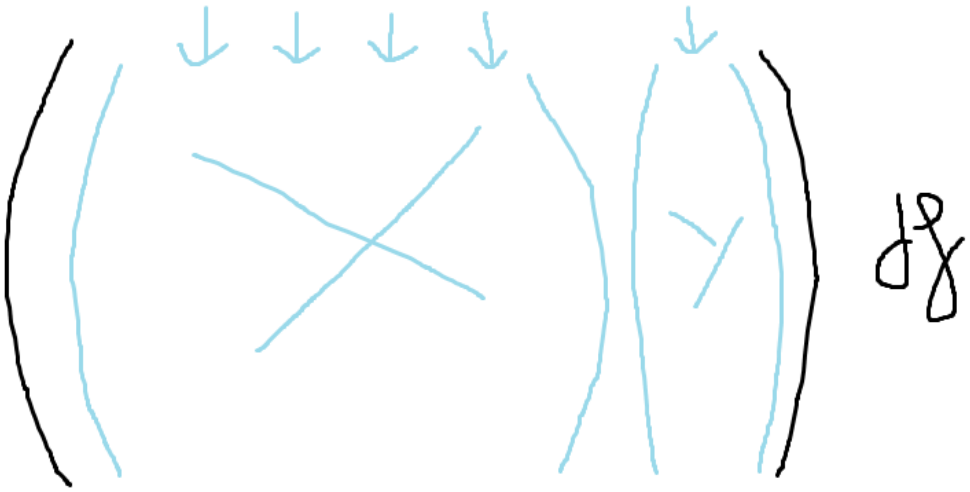
- una columna tenga valores en intervalo [0,100]
- y otra [0,10],

y eso afecte a la calidad de las predicciones.

9 X,y

- X es “Input=entrada”, la información a proporcionar,
- y es “Output=salida”, la información a predecir

```
In [15]: display(Image('C:/Users/jmp_e/Desktop/fotos/Xy.png', width=750))
```



```
In [16]: # X, y
         X = df.drop("Species", axis=1)
         X.head()
```

Out[16]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

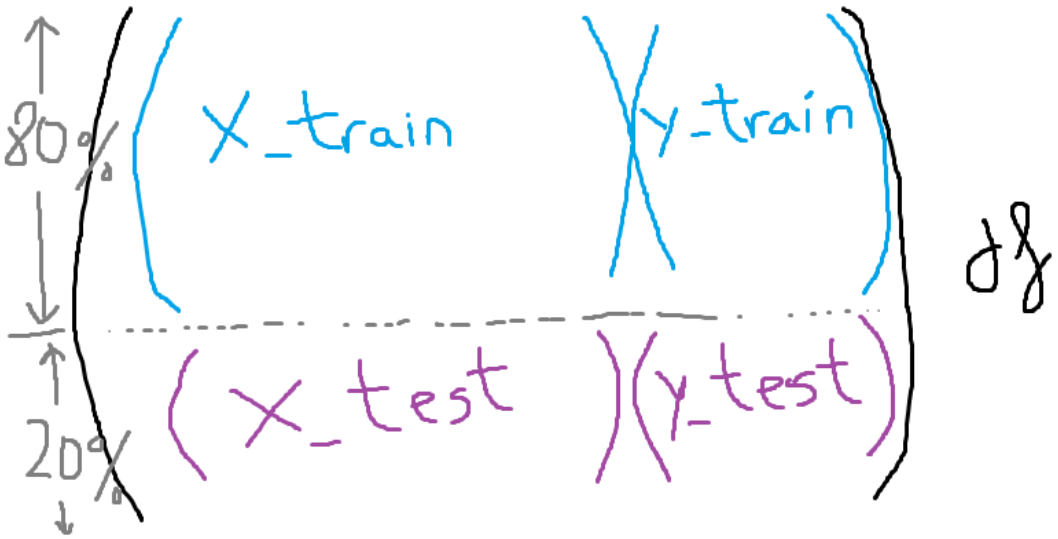
```
In [17]: y = df["Species"]
         y.head()
```

```
Out[17]: 0    0
         1    0
         2    0
         3    0
         4    0
         Name: Species, dtype: int64
```

10 Divido en Entrenamiento y prueba

```
In [18]: # separo:
         # 80% para train (entrenamiento): 80% de 150 = 120
         # 20% para test  (prueba)       : 20% de 150 = 30

         # test_size=0.80 indica el 80%
         # random_state=0 sirve para obtener siempre el mismo resultado
```

```
In [19]: display(Image('C:/Users/jmp_e/Desktop/fotos/traintest.png', width=750))
```




```
In [20]: X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.80,
                                                             random_state=0)
```

10.1 Nos fijamos en: X_train y en X_test

```
In [21]: print(df.head())
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0            5.1           3.5            1.4           0.2        0
1            4.9           3.0            1.4           0.2        0
2            4.7           3.2            1.3           0.2        0
3            4.6           3.1            1.5           0.2        0
4            5.0           3.6            1.4           0.2        0
```

```
In [22]: # Vemos que se han tomado el 80% de forma aleatoria.
         # o (pseudo)aleatoria..(no tiene importancia esto ahora)
         print(X_train.head())
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
14             5.8           4.0            1.2           0.2
122            7.7           2.8            6.7           2.0
19             5.1           3.8            1.5           0.3
29             4.7           3.2            1.6           0.2
130            7.4           2.8            6.1           1.9
```

```
In [23]: print(df.tail())
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
145            6.7           3.0            5.2           2.3        2
146            6.3           2.5            5.0           1.9        2
147            6.5           3.0            5.2           2.0        2
148            6.2           3.4            5.4           2.3        2
149            5.9           3.0            5.1           1.8        2
```

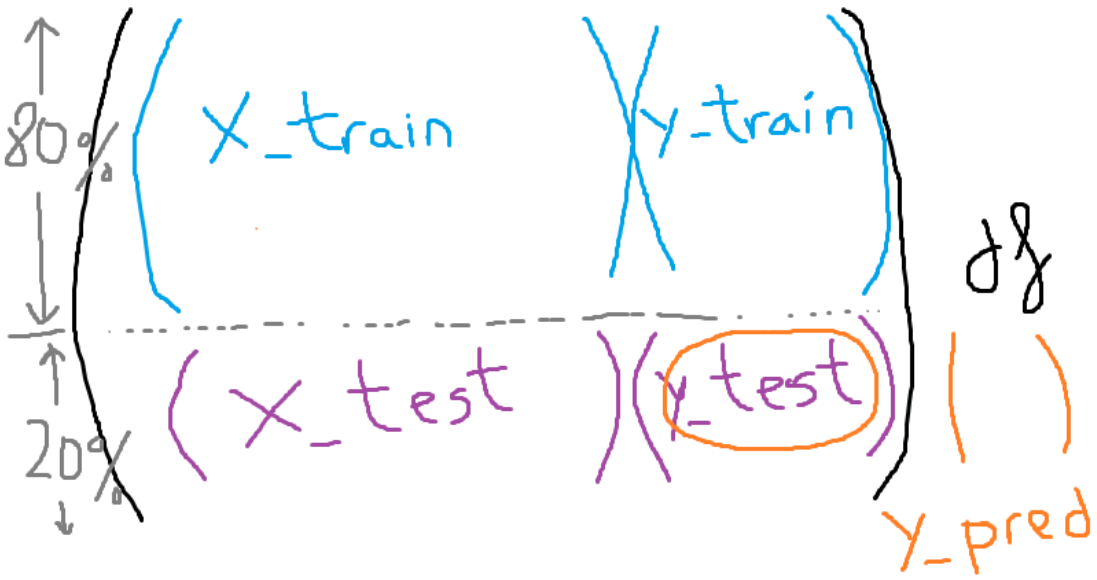
```
In [24]: print(X_test.tail())
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
57             4.9           2.4            3.3           1.0
131            7.9           3.8            6.4           2.0
65             6.7           3.1            4.4           1.4
32             5.2           4.1            1.5           0.1
138            6.0           3.0            4.8           1.8
```

```
In [25]: # NOTA:

         # En aquellos casos que hubiera problema al entrenar con los algoritmos:

         # 1 típica forma de solucionarlo es:

         # X_train = X_train.values
         # y_train = y_train.values
         # X_test = X_test.values
         # y_test = y_test.values
```

11 MACHINE LEARNING Y ALGORITMOS

```
In [26]: display(Image('C:/Users/jmp_e/Desktop/fotos/pred.png', width=750))
```



PASOS tras decidir nuestro clasificador
(DecisionTreeClassifier en este caso)
-1- ENTRENO CON: **"X_train, y_train"**
-2- LE PROPORCIONO ENTONCES LOS DATOS DE **X_test**
con la idea de que sea capaz de predecir por sí solo y_test
(en base al aprendizaje que adquirió en X_train, y_train)
y dado que para ciertos valores de "X" sale una "y" concreta..
ejemplo:
SepalLengthCm - SepalWidthCm - PetalLengthCm - PetalWidthCm - Species

```
5.1      -      3.5      -      1.4      -      0.2      -  Iris-setosa
```

Lo guarda en y_pred (y_predecida)- recordando que tratamos de predecir y_test.
(QUE YA SABEMOS LO QUE VALÍA)
-3- Analizo la fiabilidad del modelo (accuracy)
Que símplemente testea cuántos "aciertos" tiene nuestra predicción.

11.1 clasificador en este caso es: DecisionTreeClassifier( )

(ya importado en parte superior)

```
In [27]: clf = DecisionTreeClassifier()
```

11.2 Entreno con los datos de entrenamiento

```
In [28]: clf.fit(X_train, y_train)
```

```
Out[28]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

11.3 Trato de obtener los datos de y_test

y lo almaceno en y_pred (y_predecida)

```
In [29]: y_pred = clf.predict(X_test)
         y_pred
```

```
Out[29]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 1, 0, 0, 1, 1, 0, 2, 1, 0, 1, 2, 1, 0, 2, 1, 1, 2, 0, 2, 0,
                0, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 0, 2, 1, 1, 1,
                1, 2, 0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 0, 1, 2, 1, 0, 2, 2, 2, 2, 0,
                0, 2, 2, 0, 2, 0, 1, 2, 0, 0, 2, 0, 0, 0, 1, 2, 2, 0, 0, 0, 1, 1,
                0, 0, 1, 0, 2, 1, 2, 1, 0, 1], dtype=int64)
```

11.4 evaluacion del modelo - 1 posible forma

```
In [30]: # accuracy es precisión (acc)

         acc = accuracy_score(y_test, y_pred)
         acc
```

```
Out[30]: 0.9166666666666666
```

11.5 OTRA FORMA DE OBTENER "ACC"

```
In [31]: print(np.array(y_test.tolist()))
```

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0
 1 0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0
 0 1 0 2 1 2 1 0 2]
```

```
In [32]: print(np.array(y_pred.tolist()))
```

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 2 1 0 1 2 1 0
 2 1 1 2 0 2 0 0 1 2 2 1 2 1 2 1 1 1 2 1 1 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0
 1 0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 1 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0
 0 1 0 2 1 2 1 0 1]
```

£NOS FIJAMOS? Casi todos están bien predecidos !!!!

```
In [33]: y_test = y_test.tolist()
         y_pred = y_pred.tolist()

In [34]: # creamos un contador para averiguar cuantas fueron correctas.
         # lo inicializamos a 0.
         aciertos = 0

         # lo que hacemos es crear un bucle for
         # que va de 1 en 1 comprobando cuáles son iguales
         # Nota:
         # aciertos+=1 es equivalente a: aciertos=aciertos+1
         for i in range(len(y_test)):
             if y_test[i] == y_pred[i]:
                 aciertos += 1

         print(aciertos)
```

```
110
```

```
In [35]: aciertos_proporcion = aciertos/len(y_test)
         aciertos_proporcion

Out[35]: 0.9166666666666666

In [36]: porcentaje_aciertos = (aciertos_proporcion)*100
         porcentaje_aciertos

Out[36]: 91.66666666666666
```

que es básicamente lo que hace accuracy_score

```
In [37]: # existen más formas de evaluar el modelo (esta es 1 de las muchas que hay)

In [ ]:
```